Descriptive_Statistics

November 27, 2024

1 Distribution function

a continuous function is any function that does not have any unexpected changes in value. These abrupt or unexpected changes are referred to as discontinuities.

The PDF can be defined in terms of a continuous function, in other words, for any continuous function, the PDF is the probability that the variate has the value of x.

Probability mass function (PMF) is a function is associated with discrete random variables rather than continuous random variables.

The probability distribution or probability function of a discrete random variable is a list of probabilities linked to each of its attainable values.

1.1 Uniform distribution

The uniform probability distribution function of any continuous uniform distribution is given by the following equation:

$$f(x) = \begin{cases} \frac{1}{b-1} \text{ for a <= x <= b,} \\ \text{0 for x < a or x > b} \end{cases}$$

1.2 Normal distribution

Normal distribution, or Gaussian distribution, is a function that distributes the list of random variables in a graph that is shaped like a symmetrical bell. Well, a normal distribution has a density curve that is symmetrical about its mean, with its spread typically defined by its standard deviation. It has two parameters – the mean and the standard deviation. The fact that the normal distribution is principally based on the central limit theorem makes it relevant. If the size of all possible samples in a population is n, and the mean is and the variance 2 , then the distribution approaches a normal distribution. Mathematically, it is given as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\Pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad X \sim N(\mu, \sigma^2)$$

1.3 Exponential distribution

A process in which some events occur continuously and independently at a constant average rate is referred to as a Poisson point process. The exponential distribution describes the time between events in such a Poisson point process, and the probability density function of the exponential distribution is given as follows:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0 \end{cases}$$

1.4 Binomial distribution

Binomial distribution, as the name suggests, has only two possible outcomes, success or failure. The outcomes do not need to be equally likely and each trial is independent of the other.

1.5 Cumulative distribution function

The cumulative distribution function (CDF) is the probability that the variable takes a value that is less than or equal to x. Mathematically, it is written as follows:

$$f(x) = P[X \leq x] = \alpha$$

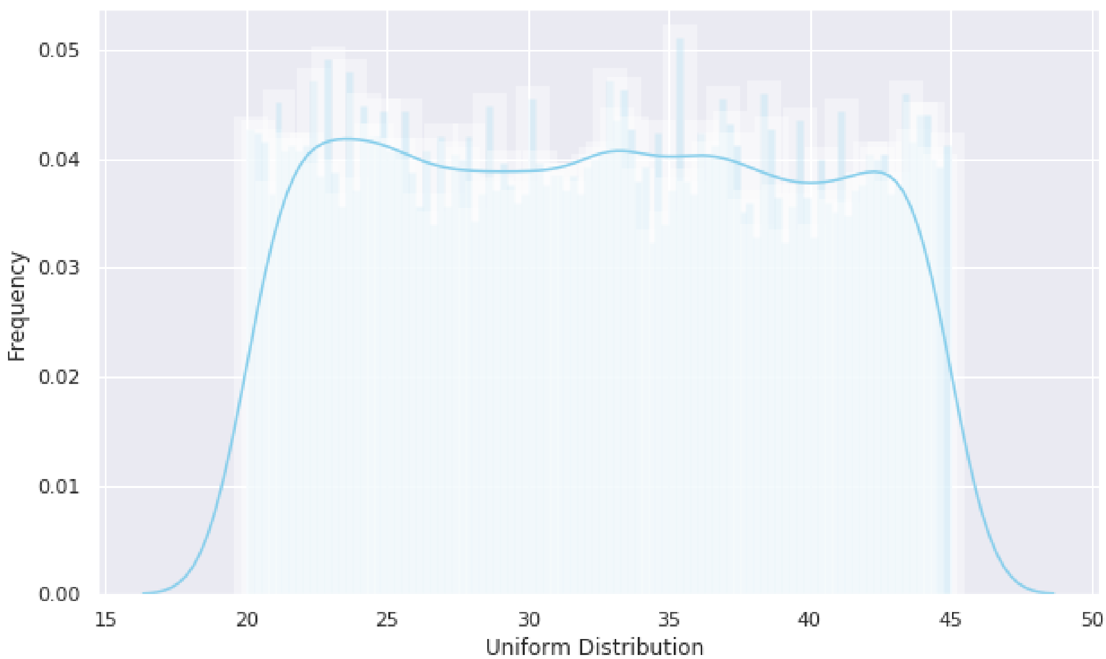
[ ]:
```
# import libraries
import pandas as pd
import numpy as np
```

[ ]:
```
import matplotlib.pyplot as plt
from IPython.display import Math, Latex
from IPython.core.display import Image
```

```python
import seaborn as sns

sns.set(color_codes=True)
sns.set(rc={'figure.figsize':(10,6)})
```

[ ]:
```python
# Uniform Distribution
from scipy.stats import uniform

number = 10000
start = 20
width = 25

uniform_data = uniform.rvs(size=number, loc=start, scale=width)
axis = sns.distplot(uniform_data, bins=100, kde=True, color='skyblue',␣
 ↪hist_kws={"linewidth": 15})
axis.set(xlabel='Uniform Distribution ', ylabel='Frequency')
```

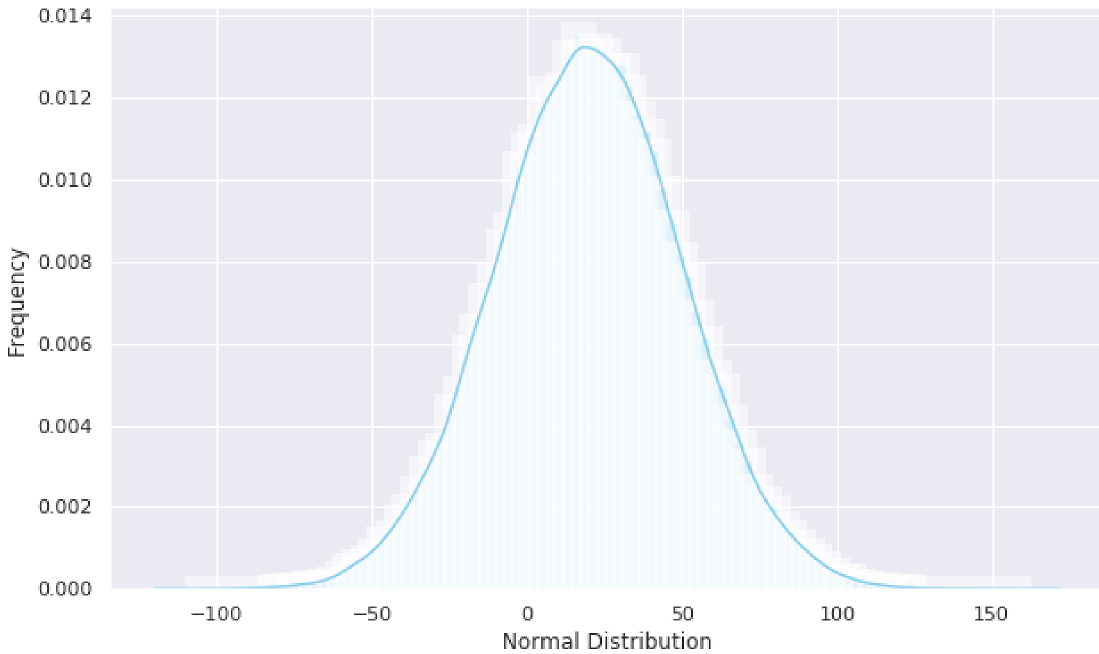
[ ]:
```
[Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Uniform Distribution ')]
```



[ ]:
```python
# Normal distribution
from scipy.stats import norm

normal_data = norm.rvs(size=90000,loc=20,scale=30)
axis = sns.distplot(normal_data, bins=100, kde=True, color='skyblue',␣
 ↪hist_kws={"linewidth": 15,'alpha':0.568})
```

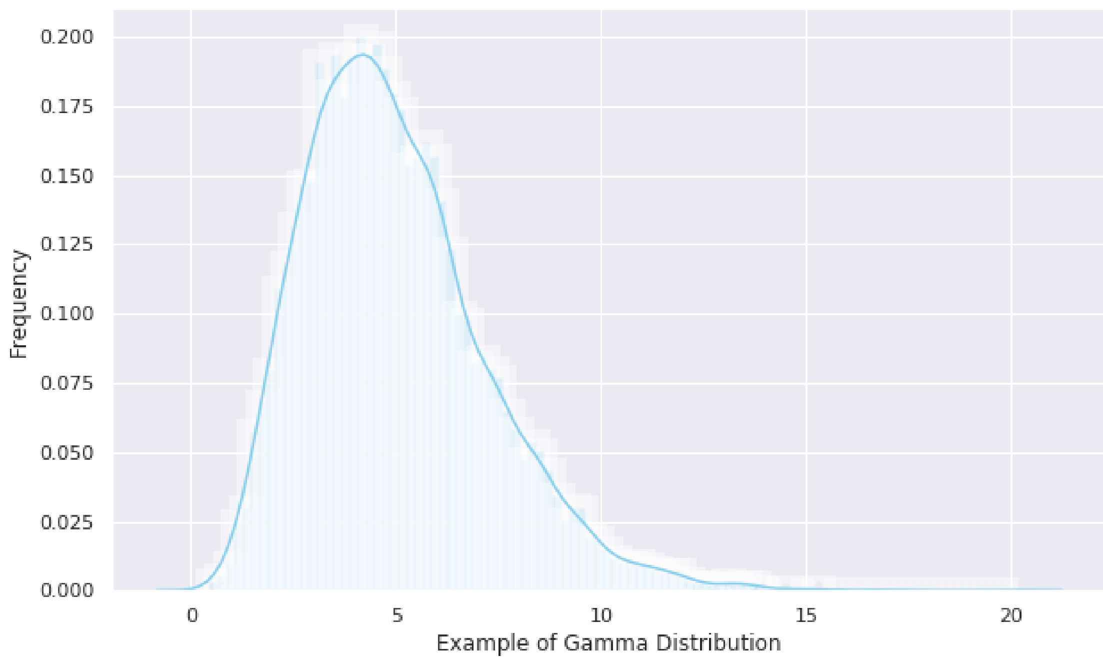
```
axis.set(xlabel='Normal Distribution', ylabel='Frequency')
```

[ ]: [Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Normal Distribution')]



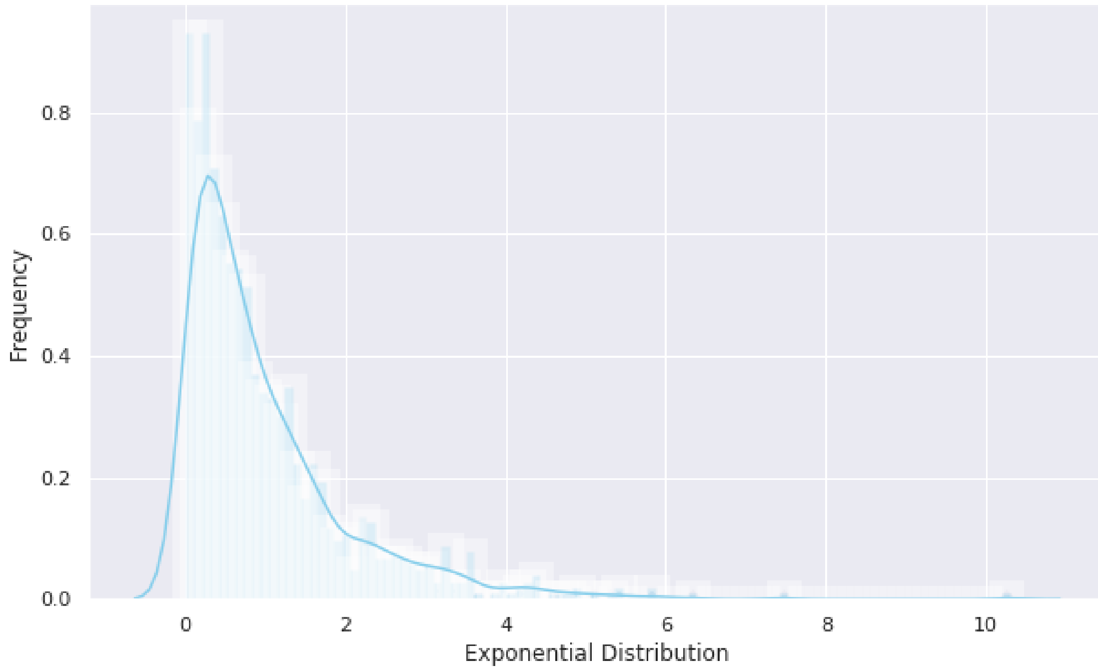
[ ]:
```
# Gamma distribution
from scipy.stats import gamma

gamma_data = gamma.rvs(a=5, size=10000)
axis = sns.distplot(gamma_data, kde=True, bins=100, color='skyblue',␣
↪hist_kws={"linewidth": 15})
axis.set(xlabel='Example of Gamma Distribution', ylabel='Frequency')
```

[ ]: [Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Example of Gamma Distribution')]



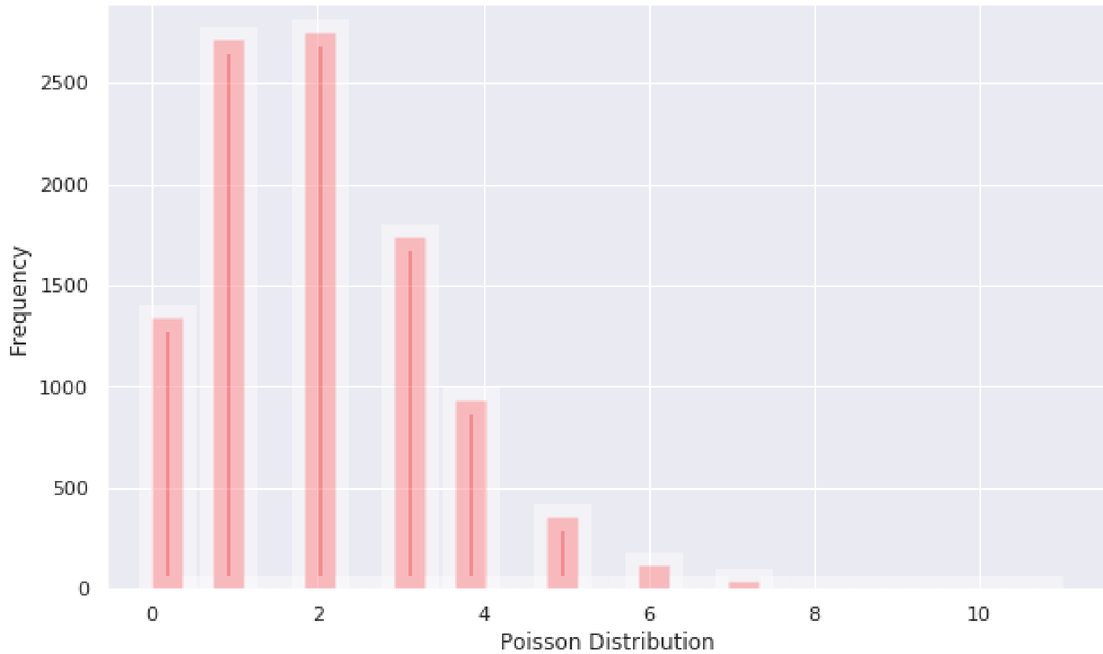
[ ]:
```
# Exponential distribution
from scipy.stats import expon

expon_data = expon.rvs(scale=1,loc=0,size=1000)
axis = sns.distplot(expon_data, kde=True, bins=100, color='skyblue',␣
 ↪hist_kws={"linewidth": 15})
axis.set(xlabel='Exponential Distribution', ylabel='Frequency')
```

[ ]:
```
[Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Exponential Distribution')]
```



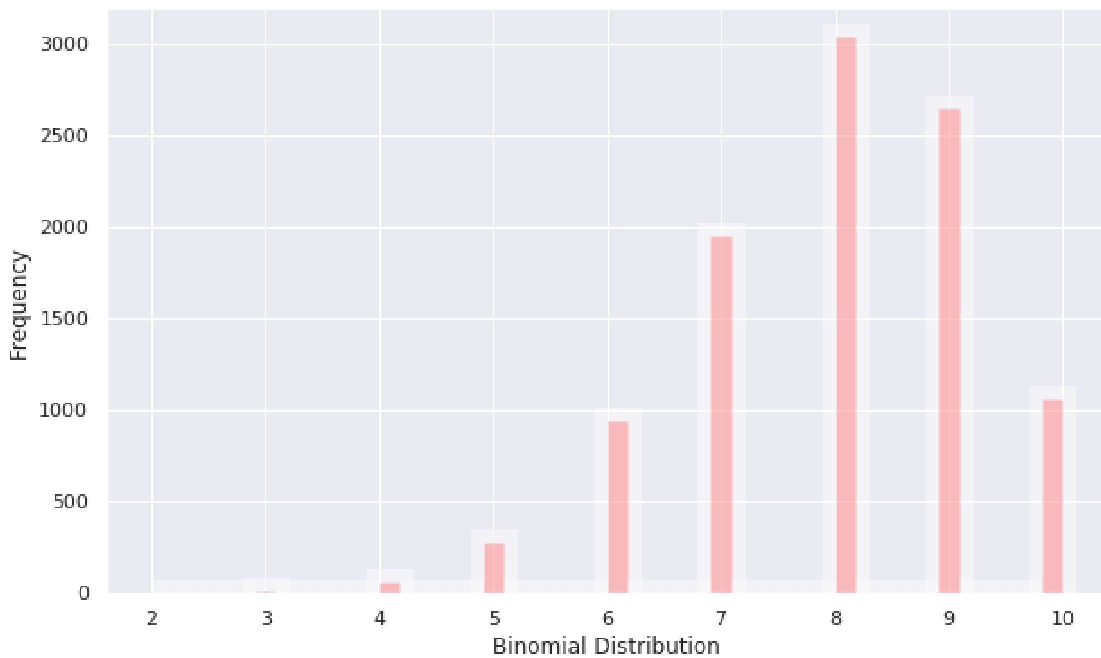
[ ]:
```
# Poisson Distribution
from scipy.stats import poisson

poisson_data = poisson.rvs(mu=2, size=10000)
axis = sns.distplot(poisson_data, bins=30, kde=False, color='red',␣
 ↪hist_kws={"linewidth": 15})
axis.set(xlabel='Poisson Distribution', ylabel='Frequency')
```

[ ]:
```
[Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Poisson Distribution')]
```



[ ]:
```python
from scipy.stats import binom

binomial_data = binom.rvs(n=10, p=0.8,size=10000)

axis = sns.distplot(binomial_data, kde=False, color='red',␣
 ↪hist_kws={"linewidth": 15})
axis.set(xlabel='Binomial Distribution', ylabel='Frequency')
```

[ ]:
```
[Text(0, 0.5, 'Frequency'), Text(0.5, 0, 'Binomial Distribution')]
```



1.6 Descriptive statistics

Descriptive statistics deals with the formulation of simple summaries of data so that they can be clearly understood. The summaries of data may be either numerical representations or visualizations with simple graphs for further understanding. Typically, such summaries help in the initial phase of statistical analysis. There are two types of descriptive statistics:

1. Measures of central tendency
2. Measures of variability (spread)

Measures of central tendency include mean, median, and mode, while measures of variability include standard deviation (or variance), the minimum and maximum values of the variables, kurtosis, and skewness.

1.7 Measures of central tendency

The measure of central tendency tends to describe the average or mean value of datasets that is supposed to provide an optimal summarization of the entire set of measurements. This value is a number that is in some way central to the set. The most common measures for analyzing the distribution frequency of data are the mean, median, and mode.

1.7.1 Mean/average

The mean, or average, is a number around which the observed continuous variables are distributed. This number estimates the value of the entire dataset. Mathematically, it is the result of the division of the sum of numbers by the number of integers in the dataset.

1.7.2 Median

Given a dataset that is sorted either in ascending or descending order, the median divides the data into two parts. The general formula for calculating the median is as follows:

$$\text{median position} = \frac{(n+1)}{2}\text{th observation}$$

Here, n is the number of items in the data. The steps involved in calculating the median are as follows: 1. Sort the numbers in either ascending or descending order. 2. If n is odd, find the (n+1)/2th term. The value corresponding to this term is the median. 3. If n is even, find the (n+1)/2th term. The median value is the average of numbers on either side of the median position.

1.7.3 Mode

The mode is the integer that appears the maximum number of times in the dataset. It happens to be the value with the highest frequency in the dataset.

2 Practice Time

[5]:
```
import pandas as pd
import numpy as np
# loading data set as Pandas dataframe
df = pd.read_csv("https://raw.githubusercontent.com/PacktPublishing/
↪hands-on-exploratory-data-analysis-with-python/master/Chapter%205/data.csv")
df.head()
```

[5]:
```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3                 ?  alfa-romero       gas        std          two
1          3                 ?  alfa-romero       gas        std          two
2          1                 ?  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  …  engine-size  \
0  convertible          rwd           front        88.6  …          130
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

  fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0        mpfi  3.47    2.68               9.0        111      5000       21
1        mpfi  3.47    2.68               9.0        111      5000       21
2        mpfi  2.68    3.47               9.0        154      5000       19
```

```
3          mpfi  3.19     3.4              10.0          102       5500         24
4          mpfi  3.19     3.4               8.0          115       5500         18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

[6]:
```
df.dtypes
```

[6]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

3 Data Cleaning

[9]:
```
# Find out the number of values which are not numeric

df['price'].str.isnumeric().value_counts()
```

[9]:
```
price
True     201
False      4
Name: count, dtype: int64
```

[10]:
```
# List out the values which are not numeric

df['price'].loc[df['price'].str.isnumeric() == False]
```

[10]:
```
9      ?
44     ?
45     ?
129    ?
Name: price, dtype: object
```

[12]:
```
#Setting the missing value to mean of price and convert the datatype to integer

price = df['price'].loc[df['price'] != '?']
pmean = price.astype(str).astype(int).mean()
df['price'] = df['price'].replace('?',pmean).astype(int)
df['price'].head()
```

[12]:
```
0    13495
1    16500
2    16500
3    13950
4    17450
Name: price, dtype: int64
```

[13]:
```
# Cleaning the horsepower losses field
df['horsepower'].str.isnumeric().value_counts()
horsepower = df['horsepower'].loc[df['horsepower'] != '?']
hpmean = horsepower.astype(str).astype(int).mean()
df['horsepower'] = df['horsepower'].replace('?',hpmean).astype(int)
df['horsepower'].head()
```

[13]:
```
0    111
1    111
2    154
3    102
4    115
```

```
Name: horsepower, dtype: int64
```

[14]:
```
# Cleaning the Normalized losses field
df[df['normalized-losses']=='?'].count()
nl=df['normalized-losses'].loc[df['normalized-losses'] !='?'].count()
nmean=nl.astype(str).astype(int).mean()
df['normalized-losses'] = df['normalized-losses'].replace('?',nmean).astype(int)
df['normalized-losses'].head()
```

[14]:
```
0    164
1    164
2    164
3    164
4    164
Name: normalized-losses, dtype: int64
```

[15]:
```
# cleaning the bore
# Find out the number of invalid value
df['bore'].loc[df['bore'] == '?']

# Replace the non-numeric value to null and convert the datatype
df['bore'] = pd.to_numeric(df['bore'],errors='coerce')
df.bore.head()
```

[15]:
```
0    3.47
1    3.47
2    2.68
3    3.19
4    3.19
Name: bore, dtype: float64
```

[16]:
```
# Cleaning the column stoke
df['stroke'] = pd.to_numeric(df['stroke'],errors='coerce')
df['stroke'].head()
```

[16]:
```
0    2.68
1    2.68
2    3.47
3    3.40
4    3.40
Name: stroke, dtype: float64
```

[17]:
```
# Cleaning the column peak-rpm
df['peak-rpm'] = pd.to_numeric(df['peak-rpm'],errors='coerce')
df['peak-rpm'].head()
```

[17]:
```
0    5000.0
1    5000.0
2    5000.0
3    5500.0
4    5500.0
Name: peak-rpm, dtype: float64
```

[18]:
```
# Cleaning the Column num-of-doors data
# remove the records which are having the value '?'
df['num-of-doors'].loc[df['num-of-doors'] == '?']
df= df[df['num-of-doors'] != '?']
df['num-of-doors'].loc[df['num-of-doors'] == '?']
```

[18]:
```
Series([], Name: num-of-doors, dtype: object)
```

[20]:
```
# it is possible to find descriptive statistics for the entire dataset at once.␣
↪Pandas provides a very useful function, df.describe, for doing so:
df.describe()
```

[20]:

	symboling	normalized-losses	wheel-base	length	width
count	203.000000	203.000000	203.000000	203.00000	203.000000
mean	0.837438	130.147783	98.781281	174.11330	65.915271
std	1.250021	35.956490	6.040994	12.33909	2.150274
min	-2.000000	65.000000	86.600000	141.10000	60.300000
25%	0.000000	101.000000	94.500000	166.55000	64.100000
50%	1.000000	128.000000	97.000000	173.20000	65.500000
75%	2.000000	164.000000	102.400000	183.30000	66.900000
max	3.000000	256.000000	120.900000	208.10000	72.300000

	height	curb-weight	engine-size	bore	stroke
count	203.000000	203.000000	203.000000	199.000000	199.000000
mean	53.731527	2557.916256	127.073892	3.330955	3.254070
std	2.442526	522.557049	41.797123	0.274054	0.318023
min	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	52.000000	2145.000000	97.000000	3.150000	3.110000
50%	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	55.500000	2943.500000	143.000000	3.590000	3.410000
max	59.800000	4066.000000	326.000000	3.940000	4.170000

	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg
count	203.000000	203.000000	201.000000	203.000000	203.000000
mean	10.093202	104.463054	5125.870647	25.172414	30.699507
std	3.888216	39.612384	479.820136	6.529812	6.874645
min	7.000000	48.000000	4150.000000	13.000000	16.000000
25%	8.600000	70.000000	4800.000000	19.000000	25.000000
50%	9.000000	95.000000	5200.000000	24.000000	30.000000
75%	9.400000	116.000000	5500.000000	30.000000	34.000000

```
max            23.000000  288.000000  6600.000000   49.000000    54.000000

              price
count    203.000000
mean   13241.911330
std     7898.957924
min     5118.000000
25%     7781.500000
50%    10595.000000
75%    16500.000000
max    45400.000000
```

Let's start by computing Measure of central tendency

[14]:
```
# get column height from df
height =df["height"]
print(height)
```

```
0      48.8
1      48.8
2      52.4
3      54.3
4      54.3
       …
200    55.5
201    55.5
202    55.5
203    55.5
204    55.5
Name: height, Length: 203, dtype: float64
```

[15]:
```
#calculate mean, median and mode of dat set height
mean = height.mean()
median =height.median()
mode = height.mode()
print(mean , median, mode)
```

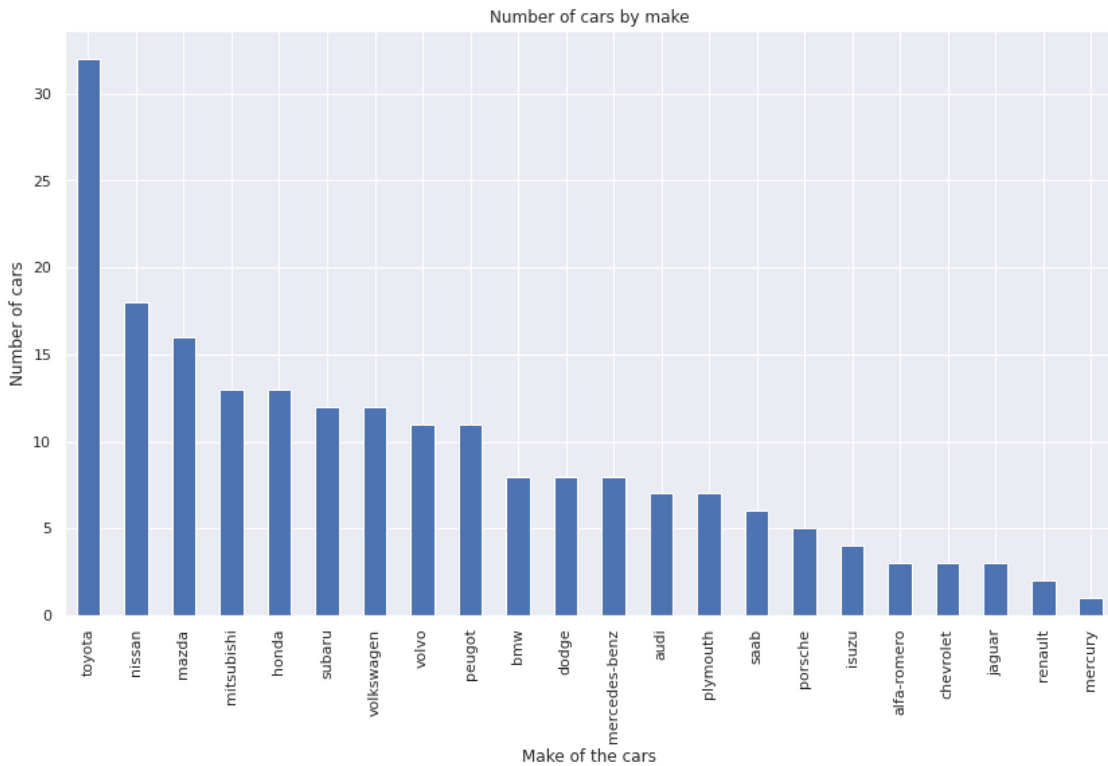
```
53.73152709359609 54.1 0    50.8
dtype: float64
```

[19]:
```
#we can understand that the average height of the cars is around 53.766 and␣
↪that there are a lot of cars whose mode value is 50.8.
```

For categorical variables which has discrite values we can summarize the categorical data is by using the function value_counts().

In the case of categorical variables that have discrete values, we can summarize the categorical data by using the value_counts() function.

[16]:
```
import matplotlib.pyplot as plt

df.make.value_counts().nlargest(30).plot(kind='bar', figsize=(14,8))
plt.title("Number of cars by make")
plt.ylabel('Number of cars')
plt.xlabel('Make of the cars');
```



3.1 Measures of dispersion

Also known as a measure of variability. It is used to describe the variability in a dataset, which can be a sample or population. It is usually used in conjunction with a measure of central tendency, to provide an overall description of a set of data. A measure of dispersion/variability/spread gives us an idea of how well the central tendency represents the data. If we are analyzing the dataset closely, sometimes, the mean/average might not be the best representation of the data because it will vary when there are large variations between the data. In such a case, a measure of dispersion will represent the variability in a dataset much more accurately

[17]:
```
#summarize categories of drive-wheels
drive_wheels_count =df["drive-wheels"].value_counts()
print(drive_wheels_count)
```

```
fwd    118
rwd     76
```

```
4wd      9
Name: drive-wheels, dtype: int64
```

3.2 Standard deviation

In simple language, the standard deviation is the average/mean of the difference between each value in the dataset with its average/mean; that is, how data is spread out from the mean. If the standard deviation of the dataset is low, then the data points tend to be close to the mean of the dataset, otherwise, the data points are spread out over a wider range of values.

[18]:
```
#standard variance of data set using std() function
std_dev =df.std()
print(std_dev)
# standard variance of the specific column
sv_height=df.loc[:,"height"].std()
print(sv_height)
```

```
symboling               1.250021
normalized-losses      35.956490
wheel-base              6.040994
length                 12.339090
width                   2.150274
height                  2.442526
curb-weight           522.557049
engine-size            41.797123
bore                    0.274054
stroke                  0.318023
compression-ratio       3.888216
horsepower             39.612384
peak-rpm              479.820136
city-mpg                6.529812
highway-mpg             6.874645
price                7898.957924
dtype: float64
2.442525704031867
```

4 Measure of variance

Variance is the square of the average/mean of the difference between each value in the dataset with its average/mean; that is, it is the square of standard deviation.

[20]:
```
# variance of data set using var() function
variance=df.var()
print(variance)
# variance of the specific column
var_height=df.loc[:,"height"].var()
print(var_height)
```

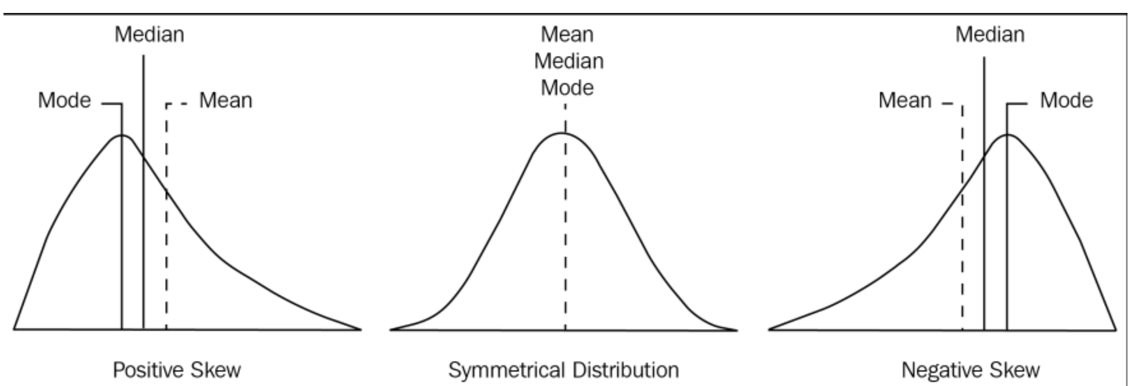
```
symboling            1.562552e+00
normalized-losses    1.292869e+03
wheel-base           3.649361e+01
length               1.522531e+02
width                4.623677e+00
height               5.965932e+00
curb-weight          2.730659e+05
engine-size          1.746999e+03
bore                 7.510565e-02
stroke               1.011384e-01
compression-ratio    1.511822e+01
horsepower           1.569141e+03
peak-rpm             2.302274e+05
city-mpg             4.263844e+01
highway-mpg          4.726074e+01
price                6.239354e+07
dtype: float64
5.965931814856368
```

[21]:
```
df.loc[:,"height"].var()
```

[21]:
```
5.965931814856368
```

4.1 Skewness

In probability theory and statistics, skewness is a measure of the asymmetry of the variable in the dataset about its mean. The skewness value can be positive or negative, or undefined. The skewness value tells us whether the data is skewed or symmetric.



1. The graph on the right-hand side has a tail that is longer than the tail on the right-hand side. This indicates that the distribution of the data is skewed to the left. If you select any point in the left-hand longer tail, the mean is less than the mode. This condition is referred to as negative skewness.

2. The graph on the left-hand side has a tail that is longer on the right-hand side. If you select any point on the right-hand tail, the mean value is greater than the mode. This condition is referred to as positive skewness.

3. The graph in the middle has a right-hand tail that is the same as the left-hand tail. This condition is referred to as a symmetrical condition.

[22]:
```
df.skew()
```

[22]:
```
symboling             0.204275
normalized-losses     0.209007
wheel-base            1.041170
length                0.154086
width                 0.900685
height                0.064134
curb-weight           0.668942
engine-size           1.934993
bore                  0.013419
stroke               -0.669515
compression-ratio     2.682640
horsepower            1.391224
peak-rpm              0.073094
city-mpg              0.673533
highway-mpg           0.549104
price                 1.812335
dtype: float64
```

[23]:
```
# skewness of the specific column
df.loc[:,"height"].skew()
```

[23]:
```
0.06413448813322854
```

4.2 Kurtosis

kurtosis is a statistical measure that illustrates how heavily the tails of distribution differ from those of a normal distribution. This technique can identify whether a given distribution contains extreme values.

It is the measure of outlier presence in a given distribution. Both high and low kurtosis are an indicator that data needs further investigation. The higher the kurtosis, the higher the outliers.

4.2.1 Types of kurtosis

There are three types of kurtosis—mesokurtic, leptokurtic, and platykurtic.

1. Mesokurtic: If any dataset follows a normal distribution, it follows a mesokurtic distribution. It has kurtosis around 0.
2. Leptokurtic: In this case, the distribution has kurtosis greater than 3 and the fat tails indicate that the distribution produces more outliers.
3. Platykurtic: In this case, the distribution has negative kurtosis and the tails are very thin compared to the normal distribution.

[24]:
```
# Kurtosis of data in data using skew() function
kurtosis =df.kurt()
print(kurtosis)

# Kurtosis of the specific column
sk_height=df.loc[:,"height"].kurt()
print(sk_height)
```

```
symboling           -0.691709
normalized-losses   -0.471235
wheel-base           0.986065
length              -0.075680
width                0.687375
height              -0.429298
curb-weight         -0.069648
engine-size          5.233661
bore                -0.830965
stroke               2.030592
compression-ratio    5.643878
horsepower           2.646625
peak-rpm             0.068155
city-mpg             0.624470
highway-mpg          0.479323
price                3.287412
dtype: float64
-0.4292976016374439
```

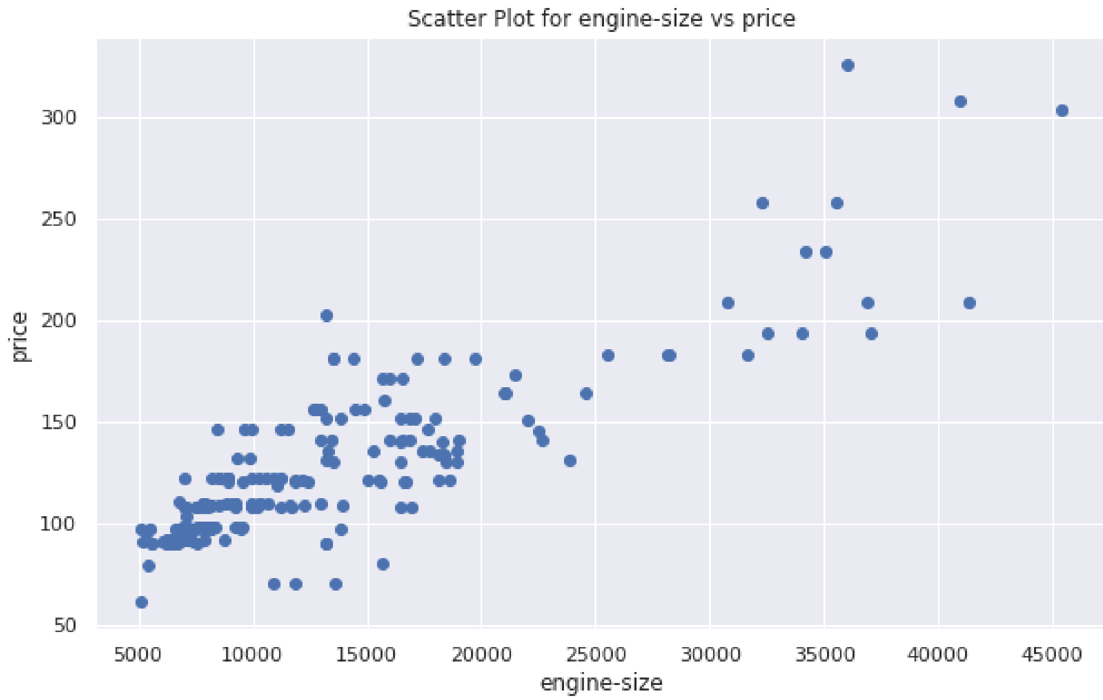
[21]:
```
height = df["height"]
percentile = np.percentile(height, 50,)
print(percentile)
```

```
54.1
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
plt.rcParams['figure.figsize'] = (10, 6)
```

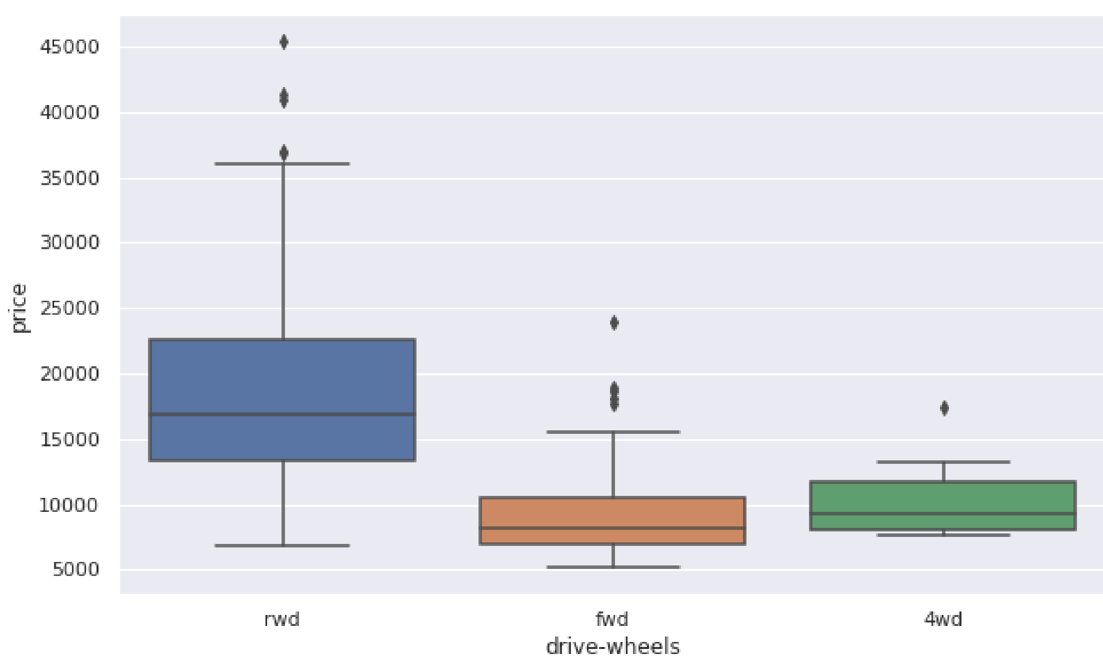
[26]:
```
# plot the relationship between "engine-size" and "price"
plt.scatter(df["price"], df["engine-size"])
plt.title("Scatter Plot for engine-size vs price")
plt.xlabel("engine-size")
plt.ylabel("price")
```

[26]:
```
Text(0, 0.5, 'price')
```




```
[27]: #boxplot to visualize the distribution of "price" with types of "drive-wheels"
      sns.boxplot(x="drive-wheels", y="price",data=df)
```

```
[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4c7d9c9dd8>
```



[28]:
```
type(df.price[0])
```

[28]:
```
numpy.int64
```

4.3 Percentiles

Percentiles measure the percentage of values in any dataset that lie below a certain value. In order to calculate percentiles, we need to make sure our list is sorted. An example would be if you were to say that the 80th percentile of data is 130: then what does that mean? Well, it simply means that 80% of the values lie below 130. Pretty easy, right? We will use the following formula for this:

$$\text{The formula for calculating percentile of X} = \frac{\text{Number of values less than X}}{\text{Total number of observations}} * 100$$

5 Calculating percentiles

[29]:
```
# calculating 30th percentile of heights in dataset
height = df["height"]
percentile = np.percentile(height, 50,)
print(percentile)
```

```
54.1
```

5.0.1 Quartiles

Given a dataset sorted in ascending order, quartiles are the values that split the given dataset into quarters. Quartiles refer to the three data points that divide the given dataset into four equal parts, such that each split makes 25% of the dataset. In terms of percentiles, the 25th percentile is referred to as the first quartile (Q1), the 50th percentile is referred to as the second quartile (Q2), and the 75th percentile is referred to as the third quartile (Q3). Based on the quartile, there is another measure called inter-quartile range that also measures the variability in the dataset. It is defined as follows:

I other words: It divides the data set into four equal points.

First quartile = 25th percentile Second quartile = 50th percentile (Median) Third quartile = 75th percentile

Based on the quartile, there is a another measure called inter-quartile range that also measures the variability in the dataset. It is defined as:

IQR = Q3 - Q1

IQR is not affected by the presence of outliers.

$$IQR = Q3 - Q1$$

[30]:
```python
price = df.price.sort_values()
Q1 = np.percentile(price, 25)
Q2 = np.percentile(price, 50)
Q3 = np.percentile(price, 75)

IQR = Q3 - Q1
IQR
```

[30]:
```
8718.5
```

[31]:
```python
df["normalized-losses"].describe()
```

[31]:
```
count    203.000000
mean     130.147783
std       35.956490
min       65.000000
25%      101.000000
50%      128.000000
75%      164.000000
max      256.000000
Name: normalized-losses, dtype: float64
```

[ ]:
```python
scorePhysics = [34,35,35,35,35,35,36,36,37,37,37,37,37,38,38,38,39,39,
40,40,40,40,40,41,42,42,42,42,42,42,42,42,43,43,43,43,44,44,44,44,44,44,45,
45,45,45,45,46,46,46,46,46,46,47,47,47,47,47,47,48,48,48,48,48,49,49,49,49,
              49,49,49,49,52,52,52,53,53,53,53,53,53,53,53,54,54,
54,54,54,54,54,55,55,55,55,55,56,56,56,56,56,56,57,57,57,58,58,59,59,59,59,
59,59,59,60,60,60,60,60,60,60,61,61,61,61,61,62,62,63,63,63,63,63,64,64,64,
64,64,64,64,65,65,65,66,66,67,67,68,68,68,68,68,68,68,69,70,71,71,71,72,72,
72,72,73,73,74,75,76,76,76,76,77,77,78,79,79,80,80,81,84,84,85,85,87,87,88]

scoreLiterature = [49,49,50,51,51,52,52,52,52,53,54,54,55,55,55,55,56,
56,56,56,56,57,57,57,58,58,58,59,59,59,60,60,60,60,60,60,60,61,61,61,62,
                 62,62,62,63,63,67,67,68,68,68,68,68,68,69,69,69,69,69,69,
70,71,71,71,71,72,72,72,72,73,73,73,73,74,74,74,74,74,75,75,75,76,76,76,
                 77,77,78,78,78,79,79,79,80,80,82,83,85,88]

scoreComputer = [56,57,58,58,58,60,60,61,61,61,61,61,61,62,62,62,62,
```

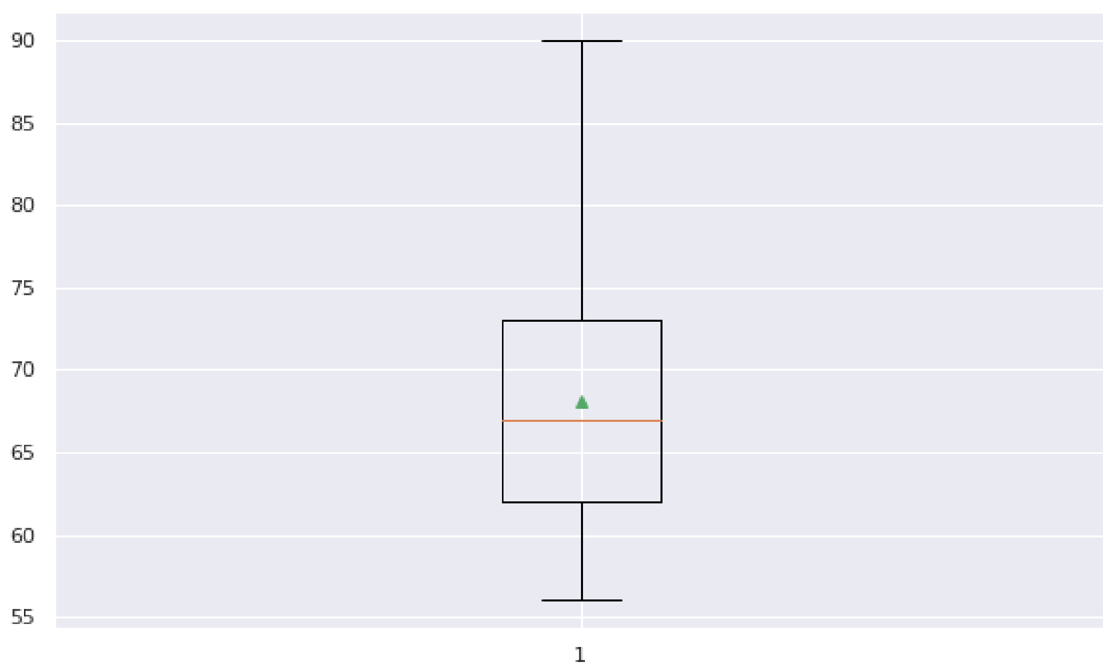
```
↪63,63,63,63,63,64,64,64,64,65,65,66,66,67,67,67,67,67,67,67,68,68,68,69,
               69,70,70,70,71,71,71,73,73,74,75,75,76,76,77,77,77,78,78,81,82,
               84,89,90]

scores=[scorePhysics, scoreLiterature, scoreComputer]
```

[33]:

```
plt.boxplot(scoreComputer, showmeans=True, whis = 99)
```

[33]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x7f4c7ac662e8>],
 'caps': [<matplotlib.lines.Line2D at 0x7f4c7ac66da0>,
  <matplotlib.lines.Line2D at 0x7f4c7ac72198>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f4c7ac72c88>],
 'means': [<matplotlib.lines.Line2D at 0x7f4c7ac72908>],
 'medians': [<matplotlib.lines.Line2D at 0x7f4c7ac72550>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7f4c7ac66630>,
  <matplotlib.lines.Line2D at 0x7f4c7ac669e8>]}
```



[35]:

```
box = plt.boxplot(scores, showmeans=True, whis=99)

plt.setp(box['boxes'][0], color='blue')
plt.setp(box['caps'][0], color='blue')
plt.setp(box['caps'][1], color='blue')
plt.setp(box['whiskers'][0], color='blue')
plt.setp(box['whiskers'][1], color='blue')
```

```
plt.setp(box['boxes'][1], color='red')
plt.setp(box['caps'][2], color='red')
plt.setp(box['caps'][3], color='red')
plt.setp(box['whiskers'][2], color='red')
plt.setp(box['whiskers'][3], color='red')

plt.ylim([20, 95])
plt.grid(True, axis='y')
plt.title('Distribution of the scores in three subjects', fontsize=18)
plt.ylabel('Total score in that subject')
plt.xticks([1,2,3], ['Physics','Literature','Computer'])


plt.show()
```